SELECTIVE INFORMATION EXTRACTION FROM NETWORK TRAFFIC TRACES BOTH ENCRYPTED AND NON-ENCRYPTED

BY

AHMAD AMRO

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

SECURITY AND INFORMATION ASSURANCE

JANUARY, 2017

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **AHMAD AMRO** under the direction of his thesis adviser and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN SECURITY AND INFORMATION ASSURANCE**.

Thesis Committee

Jan 17, 2017
Dr. Sultan Almuhammadi (Adviser)

Jan 19, 2017
Dr. Sami Zhioua (Member)

Dr. Motaz Mahmoud (Member)

Jan 19, 2017
Dr. Khalid Aljasser
Department Chairman

Dr. Salam A. Zummo
Dean of Graduate Studies

24/1/17
Date

Dedication

I dedicate this work to my father, mother, wife, sisters and brother.

My father, making you proud was my biggest motivation, I wouldn't have done it without your encouragement and support. I did it for you.

My mother, thank you for your constant prayers and support, you certainly made it easier for me. I was able to do it because of you.

My beloved wife, words are simply not enough to describe your role, you stood by me, encouraged me, enabled me to succeed. I never could have done it without you.

My dear sisters and brother, each one of you played a role in my success. I hope you're proud of your little brother.

ACKNOWLEDGMENTS

I would like to thank the KFUPM community for the best education experience, the last two and a half years have changed my life. Big thanks to my monitors, Dr. Almuhammadi, and Dr. Zhioua, from whom I've learned so much. Many thanks for Dr. Motaz Ahmed who guided me throughout my thesis. I would also like to thank my colleagues, professors, and other university staff for their help. Thank you KFUPM, thank you Saudi Arabia.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

THESIS ABSTRACT

NAME:	Ahmad Amro
TITLE OF STUDY:	Selective information extraction from network traffic traces both encrypted and non-encrypted
MAJOR FIELD:	SECURITY AND INFORMATION ASSURANCE
DATE OF DEGREE:	January, 2017

According to recent statistics, almost half the globes population are Internet users. The amount of network traffic generated by users interactions is increasing dramatically. Extracting specific types of information from network traffic has its applications in different fields in information security, especially the fields of digital forensic and web penetration testing. Analyzing large amount of network traffic for information extraction is considered challenging due to the different types of protocols implemented by different web applications, and due to the application of network traffic encryption.

In this work, different methods used for extracting different types of information from network traffic were studied. It was found that existing solutions either do not support newly adopted protocols on the web, or provide inaccurate results.

A new network data mining tool was developed as part of this work, it is called NetInfoMiner (Network Information Miner). It is capable of extracting four types of information from network traffic with the support of different types of protocols (HTTP, HTTPS, HTTP2, and SPDY). The current information extracted by NetInfoMiner are, users visited links, web credentials, session cookies, and common Facebook activities (comment, like, post, etc). A new feature that enables the logging of encryption keys into a key log file (KLF) on disk was investigated. This feature is provided by highly popular browsers. This led to the proposal of a new client side attack on TLS called DESSK (Decrypting Encrypted Sessions using Stolen Keys) as part of this thesis. The attack exploits the key logging feature through a prototype user-privileged malware called SSLKeyStealer. The malware enables the feature and steals the KLF and possibly captures network traffic then transmits these elements to a remote server. Later, the stolen elements can be used in NetInfoMiner to extract the desired information.

The work in this thesis proves through the suggested DESSK attack that the current implementation of the key logging feature by some browsers is not secure since the KLF is not encrypted and can be acquired with user-level privilege (No administrator privileges are required).

ملخص الرسالة

الاسم الكامل: أحمد وليد محمود عمرو

عنوان الرسالة: استخراج اختياري للمعلومات من سجلات بيانات الشبكات المشفرة و غير المشفرة.

التخصص: أمن معلومات

تاريخ الدرجة العلمية: يناير 2017

بينت إحصائيات جديدة ان نصف سكان الكرة الأرضية تقريبا هم من مستخدمي الإنترنت. كمية بيانات الشبكات الناتجة عن تفاعلات المستخدمين تتزايد بشكل كبير. استخراج معلومات محددة من بيانات الشبكة له تطبيقات في العديد من مجالات أمن المعلومات, خصوصا في مجال التحقيقات الإلكترونية و مجال فحص الاختراقات في مواقع الانترنت. تحليل كميات كبيرة من بيانات الشبكة لاستخراج المعلومات يعتبر من المهمات الصعبة نظرا لاختلاف انواع البروتوكولات المطبقة من مواقع الانترنت المختلفة, بالإضافة الى احتمالية ان تكون بيانات الشبكة مشفرة.

في هذه الرسالة, تم دراسة طرق متعددة لاستخراج معلومات مختلفة من بيانات الشبكة. لوحظ وجود نقص في دعم بعض البروتوكولات الجديدة و عدم دقة في نتائج الحلول الموجودة. تم تطوير أداة جديد لاستخراج المعلومات من بيانات الشبكات كجزء من هذه الرسالة, هذه الأداة قادرة على استخراج اربع انواع من المعلومات, و تدعم اربع انواع مختلفة من البرتوكولات. المعلومات اللتي يمكن استخراجها حاليا باستخدام الأداة هم, المواقع المزارة, و بيانات تسجيل الدخول (اسم المستخدم و كلمة المرور), و ملفات تعريف الارتباط (كوكيز), بالإضافة إلى بعض العمليات الشائعة على موقع الفيسبوك مثل التعليقات و الإعجاب و عمليات اخرى.

كجزء من هذه الرسالة, تم دراسة ميزة جديدة متوفرة في بعض متصفحات الانترنت و منها ما هو مشهور جدا, توفر هذه الميزة امكانية تخزين مفاتيح التشفير على القرص. بناء على هذه الميزة تم اقتراح هجوم جديد على خصوصية المستخدمين اللتي يوفرها بروتوكول التشفير. الهجوم يقوم باستغلال ميزة تخزين ملفات التشفير عن طريق برمجية خبيثة (مالوير) بصلاحيات منخفضة. هذا المالوير حين التشغيل, يقوم بتفعيل ميزة تخزين مفاتيح التشفير و يقوم بسرقة الملف اللذي يحتوي هذه المفاتيح و ارسالها للطرف المهاجم, مع وجود امكانية لتسجيل بيانات الشبكة عند الضحية. لاحقا, يتم ادخال ملف مفاتيح التشفير و بيانات الشبكة المسجلة للضحية في أداة استخراج المعلومات. ما تم عمله في هذه الرسالة يثبت ان التطبيق الحالي لميزة تخزين ملفات التشفير ليس آمنا و بصلاحيات منخفضة يمكن تحصيل ملف التشفير لكشف خصوصيات المستخدمين.

CHAPTER 1

INTRODUCTION

Internet traffic carries tremendous amount of data, streaming, files, and many other resources which can be intercepted by many parties. In particular, Internet traffic contains sensitive information such as the sequence of the users' visited links which most users prefer to keep private, login credentials which is, by far, the most commonly used access control mechanism used by web applications, session cookies which, when captured, may allow to hijack sessions, and the high-level activities performed in the social network giant "Facebook" (comments, likes, posts, etc.).

Traffic between a user and a web application (client and server) goes through a sequence of hops, including corporate gateway, routers, Internet Service provider, etc. Such setup involves a risk of eavesdropping and traffic analysis at each network hop. Uncovering specific and sensitive information from the captured traffic comes with three main challenges, namely, the huge amount of traffic, the increasing complexity of the web, and most importantly, the use of encryption. Capturing

traffic of even small local networks might generate a tremendous amount of packets. Traffic flowing through ISPs might reach 40GB per second [1]. Analysing deeply such volume of data is beyond the capabilities of most parties.

The modern web is increasingly complex, using a variety of web technologies involving highly dynamic content, extensive use of scripting languages, dynamic visual effects, and browser plugins. Consequently, a single web page might generate a large number of requests due to the fetching of several objects, automatic and continuous update of parts of the page, and advertisement content. These requests will tend to use different and obfuscated parameter naming styles and data formats. This makes the identification of specific data elements, buried in a large number of packets, a truly challenging task. The major challenge, however, is to extract useful data from encrypted traffic.

The main protection against such traffic analysis and mining attacks is to not exchange data in plain-text. However, according to an implementation survey that covers around 140,000 of the top web applications; the traffic of more than half of the industry top web applications faces the risk of exposure due to insecure or non existing implementation of encryption protocols [2]. For instance, while imdb.com is ranked 58th on Alexa most visited websites [3], it only applies encryption in the login page but not before or after, exposing users privacy (Movies and TV shows preferences and others) to any traffic sniffing attack.

1.1 Problem Statement

Extracting specific information from network traffic can be a tedious task. Existing techniques are either of limited functionality or work only with one type of traffic, either encrypted or not. A solution to work with both types and also provides sufficient summarizing and easy to deal with output report is needed. In addition to that, the encrypted traffic will require the encryption keys for it to be decrypted. So, a reliable method to get these keys is also required.

1.2 Motivation

Since 2016, Internet users have exceeded 3.3 billions [4], which directly affects the amount of traffic generated from users' interactions on the network level. According to Cisco Visual Networking Index (VNI), the amount of monthly data transferred on IP networks will reach 168 Exabyte by 2019 [5]. These statistics clearly justifies the validity of targeting network traffic for data mining, simply because everything is captured on the network level. Also, information extraction can benefit many applications. In digital forensic, any information might come in handy in digital investigations, information like, visited sites, credentials, and social networks activities. Also, in web penetration testing, the session cookies are a key information required for pen testing, getting these specific cookies from network traffic currently is not an easy and reliable option.

1.3 Research Questions

The main research question is "What types of interesting information can be extracted from this huge amount of data, and how to extract them", also "If the traffic is encrypted, are there reliable methods to decrypt the traffic". For example, if some party is interested in the login credentials of some client in a website, the only desired information is the username and password. Number of bytes in the traffic trace for the clients browsing session will most likely be in megabytes if not gigabytes, where the login credentials are only few bytes for the username and the password. So, how to get these few bytes out of the entire gigabytes of traffic, and how to make the solution general to extract the desired information from a large set of websites for large amount of users.

1.4 Scope

In this thesis, we are targeting the analysis of network traffic generated by four protocols, HTTP, HTTPS, HTTP2, and SPDY. The targeted network traffic is the one generated by direct browsing activities of Internet users, both plain and encrypted. Regarding encrypted traffic, we are not targeting VPN traffic, TOR traffic, or any encrypted traffic other than browser based SSL/TLS encrypted traffic.

Also, our suggested attack and implemented tools, target computer devices running Windows OS. Nevertheless, all implemented tools can be modified to work with devices with Linux OS's.

1.5 Thesis Organization

The organization for the remainder of the thesis is as follows. In Chapter 2, the required background for this work is provided. Beginning with a background about the SSL/TLS protocols, the handshake and key negotiation process, a quick overview for the concept of perfect forward secrecy and how it conflicts with the key logging feature. Then, a small introduction regarding the different protocols used in the WWW used for exchanging data. It also includes the prior works done that have some similarities with the work done in this thesis. Finally, a brief description of the tools used in this work is mentioned.

Chapter 3 presents the first contribution this thesis offers, a new attack against user privacy called DESSK. The chapter discusses the attack model, attack elements, and the implementation of the attack.

In Chapter 4, the second contribution is presented, a network data mining tool called *NetInfoMiner*. The chapter discusses the mining engines used for extracting different types of information. The implementation of *NetInfoMiner* and its four mining engines is also discussed in this chapter.

Chapter 5 presents the evaluation process followed in testing the different elements in this work. It includes the results obtained by testing each mining engine, in addition to some experiments conducted to test some special cases.

Chapter 6 provides a conclusion for the work, by summarizing the findings and contributions. Also, we discuss the possible future work made possible after the work done in this thesis.

CHAPTER 2

BACKGROUND AND RELEVANT LITERATURE

Data exchange between clients and servers in the World Wide Web (WWW) is carried across many protocols, some of them provide encryption as a security measure. Assuming that network traffic carries data which is exchanged using a properly implemented encryption protocol, a successful traffic analysis and mining attack requires the ability to decrypt past captured traffic, that is, after session termination. In addition to understanding the underlying protocol used to carry the plain traffic before encryption.

2.1 Secure Sockets Layer SSL & Transport Layer Security TLS

In this section, a brief background about SSL/TLS protocols is mentioned, in addition to a brief summary of the handshake process and key negotiation. Finally, the concept of perfect forward secrecy and its implementation contradictions is also discussed.

2.1.1 Background

Secure Sockets Layer (SSL) and Transparent layer Security (TLS) are protocols designed to protect both the integrity and the confidentiality of connections over the network [6]. Initially SSL 2.0 was the first version released to public in 1995. Quickly, in 1996 a complete re-design replaced SSL 2.0 into SSL 3.0 [7]. The first version of TLS was released in 1999 based on SSL 3.0 (RFC 2246) [8]. Few differences between TLS 1.0 and SSL 3.0, the main one is that TLS 1.0 doesn't provide backward compatibility to some unsupported algorithms. In 2006 TLS was updated to TLS 1.1 (RFC 4346) [9], then 1.2 in 2008 (RFC 5246) [10].

2.1.2 Handshake and Key Negotiation

When a secure connection is required to be established (i.e. access an HTTPS site), the client (usually a browser application) must perform a negotiation with the server side (web application) to establish some security parameters that en-

ables the creation of a secure connection. This negotiation is referred to as the handshake process. Two sub-protocols are the essence of TLS, the handshake and record protocols [10]. The handshake protocol handles the establishment of common cryptographic parameters needed for encryption/decryption, in addition to server authentication and possibly clients. The record protocol describes the methods for dissecting the transferred data, applying encryption and then packaging them into what is called records. This facilitates reversing the encryption and presentation of data on the other end. [6]. A detailed description of the handshake protocol is well presented in [6]. Mainly, the handshake process includes two methods for key negotiation, key exchange using RSA algorithm, or key generation using Diffie-Hellman (DH) algorithm. In RSA, as shown in figure 2.1, the client generates a random string called "pre-master secret". It will be used to generate the cryptographic keys used for encryption/decryption by both sides. A crucial step is transferring that secret to the server without exposing it to eavesdroppers. So, the client encrypts it with the public key of the server then sends it. This step is meant to authenticate the sever, since only the desired server holds the correct private key. The correct server decrypts the pre-master secret and applies the agreed upon cryptographic algorithms to generate the symmetric encryption key called "master secret". Then the master secret is used to derive the session keys. The key point to remember regarding the RSA key exchange is that the server private key can be used to calculate the session keys.

Differently in DH key generation method, the private key of the server is not

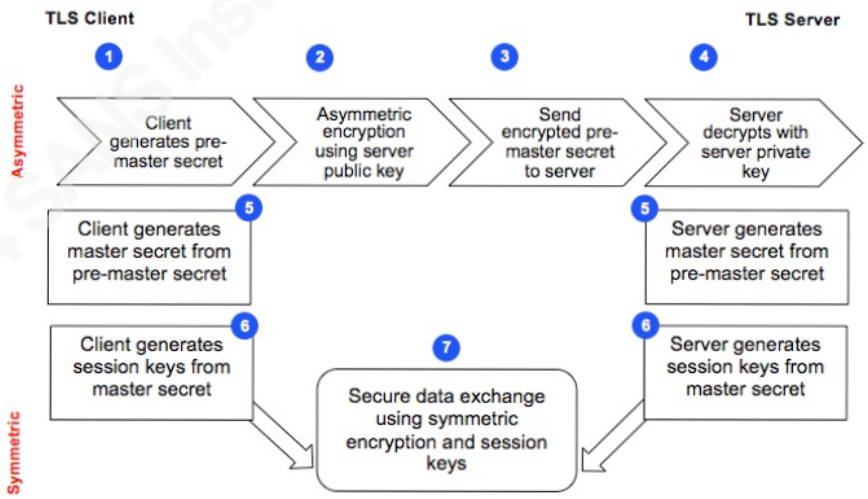

Figure 2.1: RSA key exchange method

needed in the generation of the session keys. The steps followed in the DH key generation is shown in figure 2.2. Initially, the client generates some random (ephemeral) DH components both private ones and public ones. Then, the client sends the public components to the server. On the server side, the sever components are generated and transmitted to the client. After both sides have the required components, the master secret is calculated on both sides and used to generate subsequent session keys.

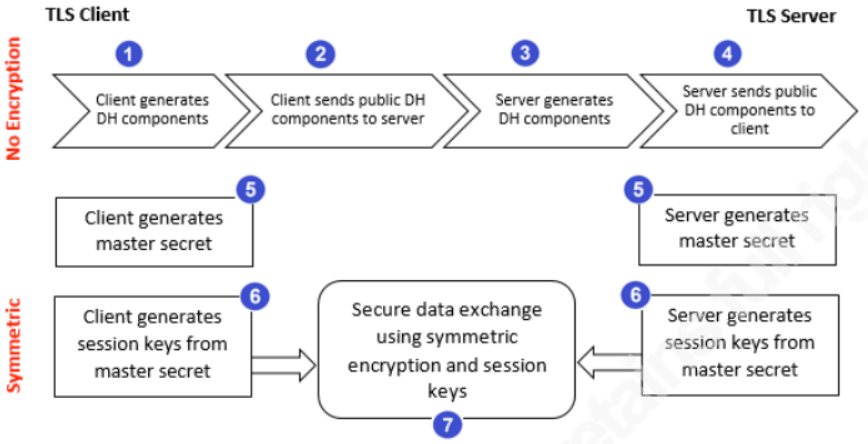

Figure 2.2: Diffie-Hellman key generation method

2.1.3 Perfect Forward Secrecy PFS

When an SSL/TLS session is created, both the client and the server are able to encrypt/decrypt exchanged content using the established symmetric key (master key or session key). Assuming that a party could capture encrypted network traffic and the encryption key has been refreshed. Basically, SSL/TLS guarantees that an attacker cannot decrypt past secure connection. This property is called Perfect Forward Secrecy (PFS) and it means that previously negotiated session keys cannot be recalculated from the client side after the session is terminated, because it requires a random (ephemeral) string and other secret components that no client is supposed to maintain. However, the private key of the server in some cases can be used to recalculate the session key, that is the case if an RSA key exchange method is used. If a Diffe-Hellman key generation method is used, the servers private key is useless in decrypting SSL/TLS sessions since it is only used to authenticate the server rather than encryption [6].

2.1.4 NSS and PEM Key Log Files

Conflicting with PFS, some browsers enable the use of "Session Resumption" to speed up the handshake process, this includes caching the negotiated SSL/TLS session keys in the browser memory or even logging them on disk [11, 12]. Browsers accounting for almost 73% of browsers market share [13], including Firefox, Chrome, Opera, Torch, and other chromium based browsers can be forced to log SSL/TLS session keys to an NSS formatted key log file (KLF) on disk if the envi-

ronment variable SSLKEYLOGFILE is set. The NSS key log format is described in details in [14]. Basically, each line in the KLF includes three columns, the first one describes the key negotiation method (RSA or non-RSA). The second column in case of RSA, holds the encoded encrypted pre-master secret, and the encoded client random for non-RSA, as either one exists in the network traffic trace. The second column is used by *Wireshark* to search the detected value in the network traffic with the matching value in the KLF. If a match is found, *Wireshark* will use the value in the third column which holds the pre-master secret (RSA) or the master secret (non-RSA) to perform the decryption using the appropriate cipher suite. This feature was meant to help analyzing encrypted SSL/TLS traffic on the client side using *Wireshark* [6].

Similarly, Some digital certificate management systems provide the option for system administrators to save the server's private key into a known format called PEM [15]. This key file format is also supported by *Wireshark* used to decrypt TLS sessions established using RSA key exchange.

2.2 WWW Protocols

There are several networking protocols used to power up web applications on the World Wide Web (WWW), the most common ones are HTTP, HTTPS, HTTP2 and SPDY.

Throughout this thesis, all the described algorithms and approaches share some similarities between HTTP, HTTP2 and SPDY. So, for simplicity we refer

to all of them as HTTP unless a distinguishing feature is needed.

A brief summary for each protocol is mentioned here:

- HTTP: The first version of the Hypertext Transfer Protocol (HTTP). An ASCII protocol used to exchange content (pages, images, etc.) between clients and servers. Although HTTP, specifically HTTP1.1 is still the most common negotiated protocol for carrying web contents, it has suffered from known limitations, such as head of line blocking, TCP handshake latency and many other [16]. HTTP protocol consists of several methods and headers to carry out its functionality. In this thesis we are interested in few of them. A brief description of some of HTTP methods and headers is mentioned here:
  - GET request method: an HTTP method for transmitting data from the client to the server. The data is transmitted as a query string in the body of the URL request (link), meaning, a request transmitted using a GET method can be fully seen in the browser an is considered "bookmark-able".
  - POST request method: same goal as GET method, but instead, the data sent from the client to the server is not publicly shown in the link and cannot be bookmark-ed by the browser. Convenient for transmitting sensitive information such as passwords.
  - HTTP Referer Header: a misspelling of referrer. It is a part of some HTTP requests. It indicates the address (link) of the web resource that initiated the request.

  - HTTP Full URI: similar to the referer header, but contains the full address (link) of the requested resource. It includes the protocol name (HTTP or HTTPS), the host domain (www.X.com), the full server path to the resource (/server_folder_1/server_folder_2/.../page.php), and the parameters sent to the resource which is called a query string(key1=value1key2=value2...).

- HTTP2 & SPDY: Google initiated the SPDY protocol (pronounced speedy) to improve both the security and the performance of HTTP. SPDY was the basis of the second version of the Hypertext Transfer Protocol (HTTP2). There are minor differences between SPDY and HTTP2, but the main methods are the same. It is important to mention two features of SPDY/HTTP2: (a) they are binary (while HTTP is textual); and (b) they multiplex several requests and responses under a single TCP connection to reduce the overhead on servers. Figure 2.3 from [17] clearly demonstrates the core difference between HTTP and HTTP2. In HTTP each request is followed by its response, making the constructing of request/response pairs from network traffic easier than HTTP2 were requests and responses are not paired together. These features highly affect the processing of web transactions on network traffic traces [16]. Processing SPDY and HTTP2 traces requires different parsing and different concept than basic consecutive request and response protocol. Also, and since these protocols are negotiated and not fully supported by all browsers, traffic traces for retrieving the same web re-

source might be different from one client to another. This effect is discussed in Section (5.3.5).

- HTTPS: Any secure connection to a web resource can be served over HTTPS, which is basically HTTP over SSL/TLS protocols.

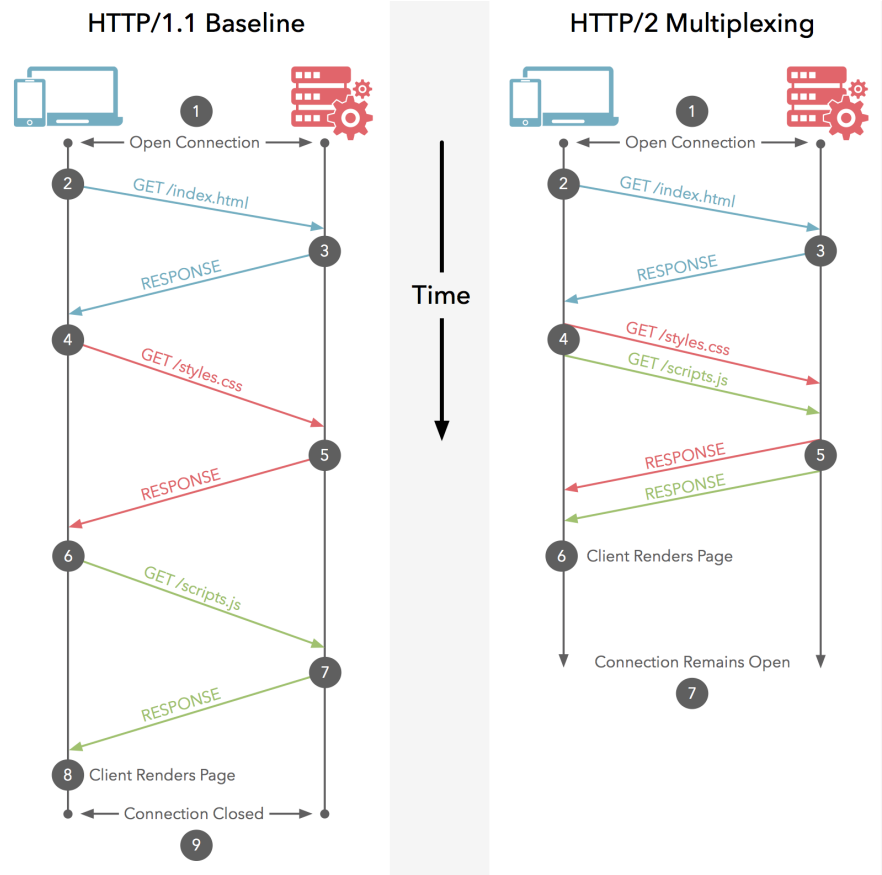

Figure 2.3: The main difference between HTTP and HTTP2

2.3 Prior Work

The work in this thesis introduces a new attack on users privacy, by attacking SSL/TLS. Also, network traffic analysis will be used to extract information about

clients. So, we divided the related works into two main streams, works related to extracting information from network traffic, and others related to different types of SSL/TLS attacks.

2.3.1 Network Traffic Information Extraction

Extracting specific information from network traffic is a well known problem in the traffic analysis field. Notable solutions in the literature target sessions [18] and files [19] reconstruction. Other works focus on detecting malware downloads in the traffic [20]. However, unlike this work, most of existing work target non-encrypted HTTP traffic only.

ClickMiner [18] is a solution created for forensic reconstruction of user browsing interactions from network traffic traces. It extracts the click path users followed in their browsing by applying an improved referrer-based click inference (RCI) approach initially introduced by Xie et al. [21]. Then, the extracted click path will be fed to a browser driver that serves these clicks using only interactions recorded in the traffic traces. NetworkMiner [19] is a commercial tool used to extract information from network traffic. It extracts files (images, certificates, and other types), HTTP parameters, cookies and many others. The free version of the tool is of limited capabilities, it doesn't support exporting results, or command line execution for automating purposes. Although the professional version provides the previous options, still, the extracted data is not effectively filtered. For instance, the tool claims to extract the credentials, but it actually extract all the parameters

of an HTTP request and displays them, even if these parameters hold other types of information like language, location, etc. Vadrevu et al. propose AMICO [20], a tool that targets live web traces for detecting malware downloads. AMICO applies what is called a provenance classifier to distinguish a malware download from a benign one based on network users downloading behaviour.

Another application for traffic analysis was proposed in [22] where the authors claimed to successfully detect visits to criminal sites by monitoring the DNS transactions recorded on network traces. The main idea behind their approach is that suspicious sites will keep changing their hosting IP address. So, the authors suggested that monitoring the relationships between the different FQDNs and the IP addresses they are hosted on will lead to detecting criminal sites.

There are some existing work that targeted encrypted traffic for information extraction. For example, in [23] since web browsers do not hide all information about the encrypted traffic, for instance, HTTP object count and sizes are often revealed. So, they claimed to successfully identify a significant fraction of a large sample of web pages based on these unconcealed information. But, a big problem with this approach, is that it can only identify static web content. Another work done in [24] where the authors applied a statistical approach to fingerprint application servers in an encrypted traffic. In their work, the authors suggested that some features like the distributions of packet sizes and inter-arrival times when fed to a machine learning classifier, can fingerprint the target server. Similar work was performed in [25] where the authors were able to fingerprint encrypted traffic

of two applications, SSH and Skype. The main idea is similar to [24] but using larger and different set of features like the packets in forward and backward direction, forward and backward packet length, etc. Also, they tested their approach using five machine learning algorithms and found that the C4.5 algorithm provides the best results. Another work in the encrypted traffic analysis track, Miller et al. [26] describe an attack on HTTPS aiming to identify web pages in top widely used web applications with accuracy reaching 89%. Their attack applies clustering techniques to detect patterns in traffic. Then they apply a group of machine learning techniques to determine similarities and detect known web pages.

The amount and quality of information extracted by analyzing the encrypted traffic might not be enough in some cases, especially if some specific information are required, like usernames, passwords, session related cookies and visited sites.

2.3.2 SSL/TLS Attacks

The availability of secure services, led people all around the world to start trusting services providers with their most private contents and information (Photos, PIN codes, SSNs, etc). Such direction led to more interest on attacking the privacy that SSL/TLS protocols were intended to protect. A quick recall to the establishment of SSL/TLS sessions. When an SSL/TLS session is created, both the client and the server are able to encrypt/decrypt its content by applying the established symmetric session key (master key). This session key in no way can be recalculated from the client side after the session is terminated, because it requires a random

string and other secret components that no client is supposed to maintain. On the other hand, the private key of the server in some cases can be used to recalculate the session key, that is the case if an RSA key exchange method was applied. If a Diffe-Hellman key generation method was applied, the servers private key is useless in decrypting TLS sessions since itis only used to authenticate the server rather than encryption [6]. At last, in the establishment of an SSL/TLS session the decision to either accept or reject the servers certificate is mostly left to the client.

If attackers are interested in the content of an SSL/TLS session, they can target either one of three possible points of attack. Either acquiring the clients session key, or the servers private key or masquerading as the server by providing a fake certificate to the client hoping for the clients acceptance. An existing categorization of SSL/TLS attacks is discussed in [27]. We suggest a simpler categorization approach. Sine an SSL/TLS session consists of three main elements, a client, a server, and a connection between them. We suggest that attacks can be categorized based on the three points of attack.

The first possible point of attack is the client. If an attacker was able to acquire a session key for some pre-recorded SSL/TLS session, that session can be decrypted. In this domain comes the attack we are proposing. Targeting the extraction of session keys to decrypt TLS sessions is not a strange approach. Taubmann et al. [28] describe a solution called TLS key extractor (TLSkex) for decrypting and analyzing TLS traffic in order to detect malicious connections. TLSKex,

records the network traffic of TLS sessions and simultaneously take a snapshot of certain parts of the main memory that most likely holds the calculated session key of an active TLS session. Their approach aims at extracting TLS master key from the virtual machine's main memory based on virtual machine introspection. In DefCon 2016, J. Kambic [12] described a solution to extract SSL/TLS session keys from memory dumps as artifacts in digital forensic to decrypt TLS sessions. Both previous works are not used to attack the privacy of users but they attack the privacy TLS provides. Our approach deviates from the two previously mentioned solutions by targeting the exported (KLF) from browsers rather than main memory and by targeting remote hosts rather than local VM hosts or a physically accessible machine.

The next possible point of attack is the server, by acquiring the servers private key. This approach will not be helpful if the key exchange method provides perfect forward secrecy, which is the case when ephemeral Diffie-Hellman key generation method is performed. In contrast, this approach is helpful when RSA key exchange is followed. This was claimed to be one of the methods N.S.A. carried to obtain unrestricted access to otherwise private information [29].

Discussing the difference between the previous two approaches is an interesting discussion to be made. A successful attack on the client side will expose sensitive information of a single client when communicating with all application servers. While a successful attack on an application server itself will expose sensitive information of all clients dealing with this server. Both client side and server side

attacks can be considered as passive SSL/TLS attacks, although they require some active steps to be performed initially, but after acquiring the necessary elements, secure SSL/TLS sessions can be exposed long after the sessions were terminated. This cannot be said regarding the third attack approach. The third family of attacks are those targeting the connection between the clients and the servers, such attacks are known as Man-in-the-Middle (MITM) attacks. Such attacks require attackers to be actively in the middle of the connection between the client and the server. MITM attacks on TLS/SSL protocols vary by their attack nature, some of them target the cryptographic aspect of the protocol like the BEAST attack which targets a vulnerability in the CBC encryption mode, others take advantage of some weaknesses in some protocol component, for instance, the CRIME attack targets TLS compression to reveal some sensitive information such as session tokens. Another example is the TIME attack which is based on the CRIME attack model but depends on the time differences of a successful and a failed guess to recover sensitive information. The latter group of MITM based attacks are described with details in [30]. Another group of MITM attacks aim to divert victims from accessing the secure HTTPS version of a certain web application to its insecure HTTP version. This group of attacks is called downgrade attacks. An example of such attack is HTTPS downgrade with ARP poisoning or JavaScript [31]. It is important to mention that HTTPS downgrade attacks are not considered as attack on SSL/TLS itself, but they are used to bypass the privacy that SSL/TLS offers. HTTPS downgrade attacks can be mitigated by the application

of HTTP Strict Transport Security (HSTS) [32]. Recently (August 2016), researchers from Microsoft reported an attack that exploits browser and operating system web proxy configuration to steal user information [33]. The attack is categorized as MITM attack and requires the installation of a self-signed certificate on the victims system. The attack is deployed over email by a malicious docx file. Upon file execution, a Javascript code will launch a group of PowerShell scripts to perform attack tasks such as snooping on HTTPS traffic. We investigated this attack and found that it is only applicable on Windows Server 2012, Windows Server 2012 R2, and Windows 8.1. We found also that to make it applicable for previous versions in Windows, more higher privileged tasks are needed [34]. Our attack model is applicable for all Windows versions that allow browsers to log session keys, in addition to Unix-based OSes.

As seen above, a wide range of MITM attacks have been proposed against SSL/TLS due to the fact that SSL/TLS protocols leave the option to either accept or reject the servers certificate to the client, which is not an easy question to be asked, especially to people with little information about the matter. An interesting point of view regarding the security of TLS is discussed in [35]. The authors discuss the browsers warning behavior and how it might affect users judgments when asked for intervention in a real attack scenario. The authors claim that browsers send the users large amount of warnings of low level risks that make the users tend to overlook sensitive events simply because "this happens a lot".

In general, and due to the fact that MITM attacks require attackers not to only

be involved in the connection between a client and a server, but also to perform some active operations during the entire TLS sessions to be able to intercept and decrypt the traffic passing through, these requirements limit the effectiveness of such attack approach compared to the other two approaches. The first limitation comes from the fact that MITM attacks in lots of cases can be detected and mitigated. A work performed in [36] where the authors presented a timing analysis approach to detect MITM attacks, the approach was built on a hypothesis that a successful MITM attack sometimes requires the generation of a fake certificate which will be reflected into a noticeable time pattern in the TLS handshake stage, a pattern not existent in a normal handshake. Also, in the same work, the authors suggested ways to detect the presence of three famous MITM attack tools, Ettercap, WebMiTM, and Cain & Abel. Each tool can be easily detected in a specific way relevant to its attack methodology. Similarly, known MITM attacks like BEAST, CRIME and others where mitigated with miner upgrades and patches [30], the same as the case of the DROWN attack, where it can be mitigated only by disabling the support for SSL v2 and updating OpenSSL to a specific newer version [37]. In addition to that, the requirement to be actively existent during active TLS sessions is not always possible, for instance, if a security incident has already happened and requires investigators to examine the encrypted traffic to extract information or evidence, in such case an active approach is useless. The last limitation can lead to describing MITM approach as Intrusive. MITM attacks will most likely leave their marks on TLS sessions. Avoiding the intrusiveness fea-

ture of any investigative approach is considered a main goal in the area of digital forensics [38].

After understanding the limitations discussed above, targeting either end of an encrypted channel might be considered a better approach. N.S.A. for instance used to follow this approach by either hacking into clients and grabbing texts before they were encrypted in many ways (keystrokes, malware, etc.) or forcing servers to hand over their private keys and sometimes stealing them [29].

2.4 Tools

In this section we provide a brief description of the tools, scripting languages, and programming languages needed to accomplish the work performed in this thesis.

1. *Wireshark (v 2.2.1)* [39]: the work performed in this thesis depends mainly on the features provided by *Wireshak* . It was used to analyze the network traffic of the different protocols; a crucial step prior to implementing the mining engines.

2. *tshark*: the command-line tool of *Wireshak*. It was used as an interface between the mining engines and the network traffic to decrypt then extract the packets with possible desired information. It's command-line nature enabled the automation of the implemented tools.

3. *editcap*: a command-line tool from *Wireshak*. It is used to split large traffic traces into smaller ones to avoid resources limitations.

4. *Windows Batch Scripting* [40]: the executive component of the implemented tools are all *.bat* files. Each file is responsible for initiating calls for *tshark* and pre-process its output then providing the pre-processed results to the mining engines.

5. *PowerShell* [41]: a tool from Windows used to automate tasks. The strong string manipulation functions provided by *PowerShell* was needed to perform some pre-processing of some *tshark* outputs.

6. *Dev-C++ (v 7.51.0)* [42]: all the mining engines were developed in C programming language using *Dev-C++* IDE.

7. *curl* [43]: a command-line tool used to to initiate requests and receive responses for different protocol types, like HTTP, FTP, SMPT, and others. It was needed in the implementation of the *SSLKeyStealer* malware.

8. *RawCap* [44]: a portable command-line tool for network traffic capture. It was needed in the implementation of the *SSLKeyStealer* malware.

9. *VirtualBox* [45]: a virtualization tool from Oracle. The testing virtual machine (VM) distributed among the test subjects was prepared and launched using *VirtualBox*.

CHAPTER 3

CLIENT-SIDE TLS ATTACK

This thesis proposes a new client side attack called DESSK (Decrypting Encrypted Sessions using Stolen Keys), the attack exploits browsers feature that enables SSL/TLS session keys logging. An attacker having the sessions key log file can use it to retrieve plain-text data from SSL/TLS encrypted traffic. The traffic can then be processed by our proposed traffic mining system to extract sensitive pieces of information, in particular, the sequence visited URLs, login credentials, session cookies, and common Facebook activities (comment, like, post, etc.). The system uses heuristics to identify obfuscated credentials in various formats. It then presents the data in a summarized and structured XML format. The DESSK stealing feature has been implemented in a user-privilege malware (not requiring administrator privilege) called SSLKeyStealer.

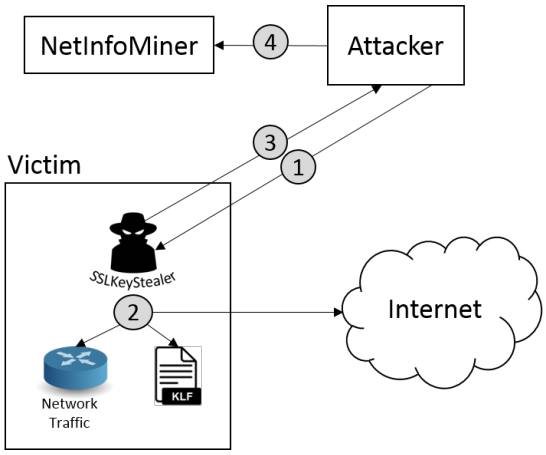

Figure 3.1: DESSK Attack Model

3.1 Attack Model

The DESSK attack model is shown in Figure 3.1. Initially the attacker sends a user-privilege malware called SSLKeyStealer to a target victim. The malware can be delivered via E-mail, Drive-by download, etc. When launched, SSLKeyStealer performs some tasks to collect the KLF and possibly the captured network traffic. When received, the stolen KLF and the captured network traffic are passed to our network traffic mining system, *NetInfoMiner*, which decrypts the encrypted network traffic using the stolen session keys. *NetInfoMiner* depends on *tshark*, the command-line tool provided by *Wireshark*, to perform the decryption.

NetInfoMiner can then extract the desired information, namely, the victim's visited links, login credentials, session cookies, and common Facebook activities. More details about *NetInfoMiner* is discussed in 4. All previous extracted information can be later used by the attacker to stage further attacks such as session hijacking on the victim.

We identified two scenarios for DESSK attack: local victim scenario and re-

mote victim scenario. In a local victim scenario, the attacker is either in the same LAN as the victim or has access to the network traffic in the path from the victim to the destination web servers (Corporate router, Gateway, ISP, etc.). In a remote victim scenario, the attacker has no direct access with the victim nor to its network traffic. In both scenarios, the victim must run SSLKeyStealer which requires user privilege (no need for administrator privilege), and the remote network must allow FTP or SMTP traffic, since SSLKeyStealer will use either one to transmit the stolen elements. The only difference is the requirement to capture network traffic. In the remote victim scenario, SSLKeyStealer is responsible for capturing the traffic on the victim's machine. This does not require higher privilege if a network sniffer tool is already pre-installed (e.g. *Wireshark* [39], Tcpdump [46], etc.). Otherwise, a privilege escalation becomes a must to use the companion portable traffic capturing tool, namely, RawCap [44].

3.2 SSLKeyStealer

A key component in the DESSK attack is the SSLKeyStealer malware. When launched, SSLKeyStealer performs the following tasks as shown in Figure 3.2:

1. Look for an environment variable called SSLKEYLOGFILE. Its value should point to the KLF.

2. If there is no such environment variable, create it.

3. Close all the supported browsers (the ones that support the exporting of

KLF).

4. If the attacker doesn't have access to network traffic, SSLKeyStealer will initiate network traffic capture.

5. On first launch of any supported browser, the KLF is created.

6. Create a scheduled task to periodically send the KLF as well as the traffic capture to the attacker (FTP or SMTP).

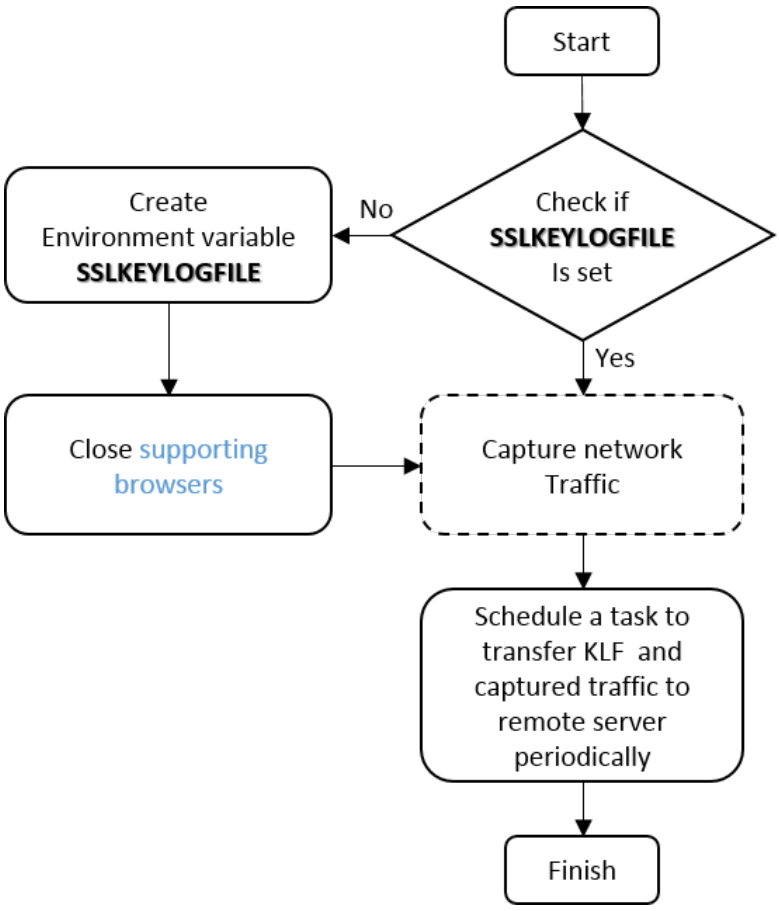

Figure 3.2: SSLKeyStealer Flowchart

3.3 Implementation

The prototype for the SSLKeyStealer malware is developed using windows batch scripts. It consists of four main components:

1. Infection vector: responsible for the handling of the environment variable SSLKEYLOGFILE and closing the supported browsers. Its functionality can be acquired using commands in a .bat script.

2. Scheduler: responsible for the creation of a scheduled task on the client side to periodically send the KLF and the captured network traffic. Its functionality can be acquired by initiating commands to the local "Task Scheduler" in Windows, or its equivalent in Linux. It forces the OS to create a task to launch the Transmitter script periodically.

3. Transmitter: responsible for the transmission of the stolen components to a remote server. Its current supported protocols are FTP and SMTP. It leverages the portable curl tool.

4. Sniffer: responsible for sniffing victims network traffic. If needed, the sniffer will search the victims machine for an existing sniffing tool and leverage it to sniff traffic. If no sniffing tool exists, the sniffer will initiate commands to the companion RawCap sniffing tool. The later commands will require the user to accept privilege escalation.

The current implementation of SSLKeyStealer works for Windows machines and tested on Windows 7, 32-bit and 64-bit platforms.

CHAPTER 4

INFORMATION EXTRACTION

Huge amounts of data is transferred by network traffic through different network components. The rapid increase of Internet users make such data both available and valuable. Since 2016, Internet users have exceeded 3.3 billions [4], which directly affects the amount of traffic generated from users' interactions on the network level. According to Cisco Visual Networking Index (VNI), the amount of monthly data transferred on IP networks will reach 168 Exabyte by 2019 [5]. Part of this data includes private high-level information such as browsing history, users' credentials, session management cookies, and other information, which are typically encoded in low-level format within the network traffic trace.

Extracting interesting information from network traffic is hard and tedious. It requires understanding of all the protocols responsible for communicating information over the network, and the criteria of generating the traffic. The process of extracting information becomes more challenging if the network traffic is encrypted. According to a TLS implementation survey [2], about 53.6% of the industry top

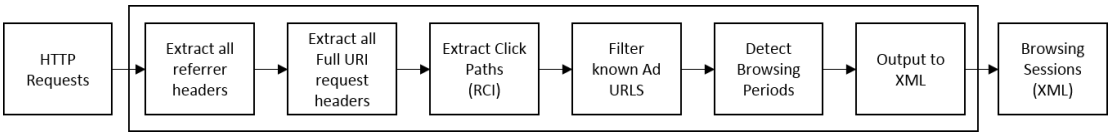

Figure 4.1: Visited Links Mining Engine

140,000 websites apply inadequate security implementation, which means either insecure or absent implementation of TLS. This indicates that targeting both encrypted and non-encrypted network traffic for information extraction is equally needed. In the work performed in this thesis, we developed a new tool called *NetInfoMiner*. A tool for extracting some high-level information from network traffic. The current information types targeted y the tool are the visited links, login events (including usernames and passwords), session cookies, and common Facebook activities (comments, likes, posts, etc.). The tool supports four protocols (HTTP, HTTPS, HTTP2, and SPDY). It is designed to process large amount of network traffic traces.

4.1 Visited Links Extraction

The first information *NetInfoMiner* targets for extraction is the visited links that clients followed in their browsing sessions. The tool follows a referer-based click inference (RCI) approach originally mentioned in [21]. To analyze encrypted HTTPS traffic, with the additional support for HTTP2 and SPDY, a different RCI is implemented by adding the idea of filtering known automatically generated advertisement requests mentioned in [18]. Figure 4.1 summarizes the new RCI implementation. The original RCI approach claims that a visited link can

be detected by matching a referer URI with a previously recorded request. The tool extracts the visited links and group them logically in a new representation suggested by us called a *browsing session*. Each *browsing session* includes a series of visited links by a specific client using a specific browser without the client being idle for more than a period of time τ. Links visited by the same client using a different browser or after being idle for more than τ are represented in a different browsing session. We observe that this representation helps understanding clients' behavior better than a list of visited links.

4.2 Credentials Extraction

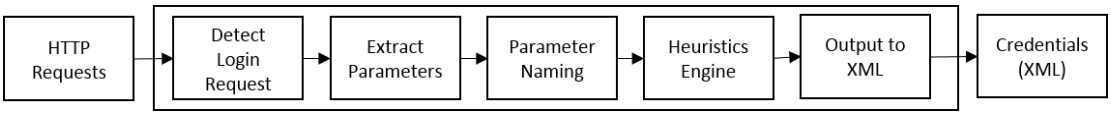

Figure 4.2: Credential Engine

The credentials information extractor is in charge of retrieving web applications credentials (usernames and passwords). The two main challenges for credentials information extraction are the large amount of traffic to be analyzed and the variety of ways web applications encode the credentials values.

A summarized sequence of steps of our credential mining methodology is shown in Figure 4.2. Assume a victim generated a set of packets of network traffic, y, in an hour, typically $|y| \simeq 53$ Million packets in a 10Mbps bandwidth [47]. Let n be the number of web applications the victim logged in to during that hour, and x be the set of the packets generated from all login operations. Let us assume that

each login operation is sent in one packet, and a typical user does not log in to more than 100 websites in an hour, that is, $|x| = n < 100$. In order to acquire such a small set x from a large set of packets, y, with $|x| \ll |y|$, a technical survey of the login operations in some web applications was conducted. The following common features were observed, they are used to identify the packets in x with high accuracy:

- The commonly used protocols for communicating credentials to web applications are HTTP, HTTP2, and SPDY. (Adopting HTTP2 has been increasing since October 2015, and adopted by the top web applications: Facebook, Google, Twitter and others [48])

- The only observed method used to transmit credentials is the "POST" method.

- The two observed "Content-type" headers used for credentials are URL-encoded form and JSON. (These types of content carry out the credentials in a structure of parameters (or keys) and their assigned values)

- The target URL of the login requests usually suggest a login page. It includes a phrase like Login, Logon, SignIn, etc.

Acquiring the packets with the above features is achieved through filtering. A number of broad *Wireshark* filters are constructed to retrieve such packets. In a perfect scenario, the resulting set of packets z should be the same as x which is not the case most of the time. A packet $p \in x$ but $p \notin z$ means that p is not

captured by the filters due to different features than the ones mentioned above, resulting in p considered as false negative. The opposite case, that is, for a packet $p \notin x$ but $p \in z$ means that the filters captured some possible login operations which are not, resulting in p counted as false positive. Furthermore, to improve the identification accuracy of login requests, he parameters names and their values will be deeply analyzed. From the technical survey it was noticed that most of the parameters carrying the usernames and passwords are self-descriptive (using meaningful parameter names). There are few cases where the username and password parameters were not self-descriptive, especially in web applications that do not use HTTPS for sending the credentials. Web applications with HTTPS will rely on encryption to obfuscate the credentials. However, non-HTTPS applications should use their own encoding to obfuscate credentials.

The identification of true username and password parameters is divided into two steps, for each packet in z, all its parameters will be first analyzed by its naming style, then by its value. For the processing of the parameter naming style, a list was prepared consisting of keywords in which username and passowrd parameters are carried across. Such list of keywords includes: username_key, uname, email, session_user, PWORD, passw , etc. If such parameter is found and it includes a value, that value is assumed to be either a username or a password depending on the parameter name. If, on the other hand, the parameter name does not match any of the above list of keywords, it will be passed to a heuristic engine. The heuristic engine uses a set $H = h_1, h_2, ..., h_r$ of low-level heuristics to compute

and assign a pair of scores (un_k, pw_k) for each parameter k indicating how k is related to the respective credential (username or password). The scoring heuristic is explained in Algorithm 1. The algorithm decides whether a given parameter k is considered as a *username*, *password*, or simply ignored. A parameter k with a score pair satisfying $max(un_k, pw_k, t) = un_k$, for some fixed threshold value t, is considered to be a username. On the other hand, if $max(un_k, pw_k, t) = pw_k$, then k is considered as a password. If neither score exceeds t, the parameter k will be ignored.

Algorithm 1 Scoring Heuristic

Input: Parameter k, Heurestic_Set H, Thresholds t
Output: $decision_k \in \{username, password, ignore\}$

$un_k = 0$
$pw_k = 0$
for $i = 1$ to $|H|$ **do**
    if $Match\_Username(k, h_i)$ **then**
        $un_k = un_k + h_i(k)$
    end if
    if $Match\_Password(k, h_i)$ **then**
        $pw_k = pw_k + h_i(k)$
    end if
end for
if $max(un_k, pw_k, t) = un_k$ **then**
    $decision_k = username$
else if $max(un_k, pw_k, t) = pw_k$ **then**
    $decision_k = password$
else
    $decision_k = ignore$
end if
Retrun $(decision_k)$

The heuristic set H in Algorithm 1 is constructed using some results collected from different sources:

1. a study performed on password enforcement policies in Alexa top 25 websites [49],
2. an experiment on the trends followed in password creation [50],
3. A study performed in this thesis of the 10 million stolen usernames and passwords data set found in [51].

The above results include the following:

- It is a trend for usernames to be in the form of emails.
- Passwords created with enforced policies have identifiable features related to the creation policy, such as having combinations of symbols, digits, upper and lower case letters.
- Out of 10 Million usernames 97.5% have 4-16 characters without including the domain name in case of emails.
- Out of 10M passwords 94.4% have between 4 and 11 characters.
- 7% of 10M passwords start with a capital letter.
- Passwords might have letters then numbers or vice versa.
- 45.5% of 10M passwords end with digits.

- 68.1% of 10M passwords and 92.3% of usernames have more alphabetical than non-alphabetical characters.

- 90% of usernames and 88% of passwords belong to set of patterns. These patterns are shown in Table 4.1, only the patterns that represent more than 1% are shown. For instance, in the 10 million passwords, a total of 11715 patterns were detected, only 8 of them represent more than 1% of the passwords. in total these 8 patterns accumulate 88.70% of all the passwords patterns.

Table 4.1: Detected usernames and passwords patterns in the 10M data set

		Usernames		Passwords	
All		6281		11715	
>1%	**Patterns**	s	43.57%	s	38.24%
		sD	22.25%	D	20.35%
		sbs	10.93%	sD	19.11%
		Cs	3.43%	Ds	4.45%
		sDs	2.57%	sDs	2.51%
		sbD	1.86%	CsD	1.71%
		CsD	1.62%	Cs	1.26%
		D	1.52%	C	1.09%
		sbsD	1.25%		
		C	1.14%		
	Total	10	90.16%	8	88.70%
C: capital letter character s: small letter character D: Digit (0-9) b: symbol					

To identify the heuristic scores of usernames and passwords, the heuristics were implemented on the 10 million stolen usernames and passwords data set. The calculated scores of all the usernames and passwords are shown in Figure 4.3. The figure shows the ranges of the calculated heuristic scores for the usernames in

the top and for passwords in the buttom. For instance, 2564495 passwords have a score between 130.32 and 140.35.

Based on the calculated scores, the ranges to identify usernames and passwords have been set and are shown in table 4.2. An overlap in the scores ranges between usernames and passwords can be noticed which will be reflected in the evaluation section 5.3.3 as a main source for false detection.

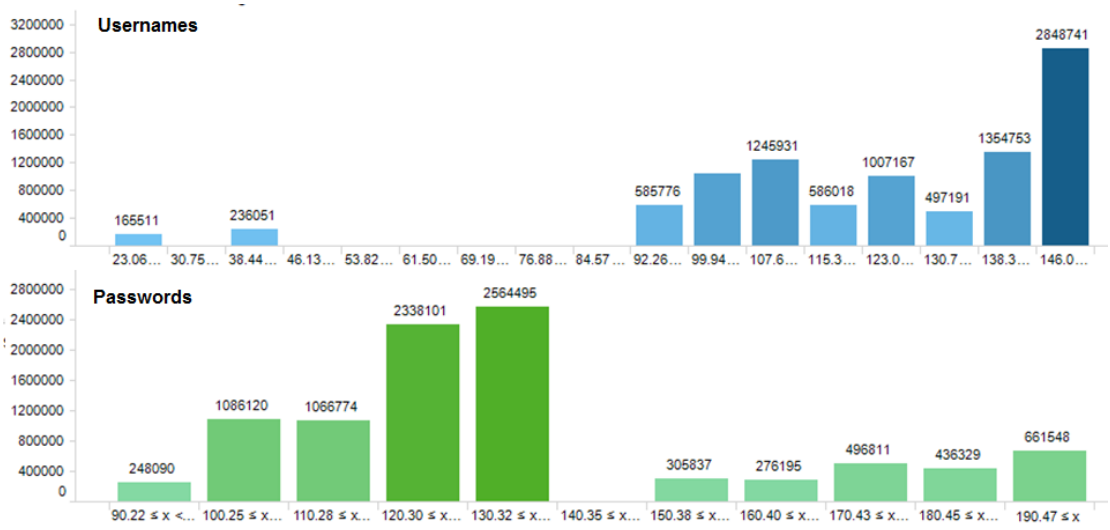

Figure 4.3: Calculated heuristic scores for the 10 million usernames and passwords

Table 4.2: Heuristic scores ranges

	From	To	From	To
Password	90.2	140.4	150.4	201
Username	92.3	153.8	-	

After detecting either a username or a password parameter, the process of building the credentials list starts. Each detected username or password is assigned to a credential entry in the credential list. Assume that a username parameter *uname* is detected with no password for a specific web application w. A credential entry c is created for w with only a username parameter. Assume that,

shortly after, a password parameter *pword* is detected for w. Then *pword* should be added to c. The previous case is a typical case for detecting credentials in web applications that implement two steps login operations, where the username and the password are entered separately, such as Yahoo and Google.

Another case might happen when a credential entry c exists for a web application w with a username parameter *uname* and a password parameter *pword* before detecting the same parameters *uname* and *pword* again but with different values. This would suggest a possible incorrect credentials. For possible forensic applications of *NetInfoMiner*, all the detected credentials are saved even if they might be incorrect, since they can be used as a proof of a login attempt. To verify whether the detected credentials yielded a successful login or not, two solutions are presented: (1) monitoring the creation of a session management cookie which might indicate a successful login operation, and (2) analyzing the HTTP response for an incorrect login. An incorrect login will result in an HTTP response with an HTML page including a phrase suggesting a failed login such as "Incorrect username or password" or other error messages, or if the login form is displayed again, since sometimes a failed login does not generate any errors.

4.3 Session Cookies Extraction

Figure 4.4: Session Cookies Engine

The Third type of information extracted by *NetInfoMiner* is session cookies. Web application servers communicate the establishment of a cookie to clients using the set-cookie header in HTTP responses. Upon successful login, the client receives a Set-Cookie header from the server with the value of the cookie that will be later included in every transaction between the client and the server in the current session. Apart from authentication, Web applications use cookies for other purposes. Hence, not all set-cookie headers contain a session cookie. To distinguish a session cookie from other variants, the naming of the newly set cookies was targeted. Famous web development frameworks can be fingerprinted using their cookie naming. For instance PHPSESSID is typically used in PHP, JSESSIONID in J2EE, CFID and CFTOKEN in ColdFusion, etc. [52]. The web framework usage statistics justify that direction, since the collected session cookie names covers more than 70% of the mostly used frameworks, for instance PHP is used in 26% of all websites, several ASP versions around 30%, J2EE with 9%, etc. [53]. A list of known session cookie names was prepared, in addition to other customized names for the session cookies implemented by some famous web applications, Facebook for instance applies three cookies to maintain user sessions, "datr", "c_use", and "xs" [54]. Any detected cookie which is set in a time frame close to a detected login attempt to the same host, and it's name matches a keyword in the prepared list, that cookie will be assumed as a session cookie.

$NetInfoMiner$'s approach for detecting session management cookies as shown

in Figure 4.4 can be summarized as follows:

- Extract all HTTP responses with Set-Cookie headers.
- Detect a cookie with a name matching the known names of session cookies (i.e. JSESSIONID).
- Filter out known non-session management cookies (i.e. _ga, __utma, __utmb, etc.).
- The remaining session cookies are grouped in cookie sets, each set represent the cookies defined between a client and a server.
- The final step is exporting the detected cookie sets into an XML structure. A sample extracted cookie sets is shown in Figure 4.9. The values of the cookies are not shown in this sample for privacy reasons.

4.4 Social Network Information Extraction

The fourth type of information extracted by *NetInfoMiner* is related to tracking some users activities on the largest social network, Facebook. The current tracked activities are, comment, reactions (like, love, etc.), add post, edit post and delete post. The methods Facebook follow to implement such activities were studied by capturing the network traffic generated by performing them and analyzing them using *Wireshark*. The summarized methodology for the social network mining engine is shown in figure 4.5. To extract only the packets with interesting activities, a group of *Wireshark* filters was constructed, the filtering scheme is based

on searching for requests with certain keywords, these keywords are the parameters names used by Facebook to implement each activity. Each captured request contains a large number of parameters needed by Facebook. In the scope of our work, we are only interested in few of them, such as, the text, the user, the privacy settings (Only me, Friends, Public, or Custom list),the time, and the activity type. After extracting the desired information, a time line of the activities will be constructed.

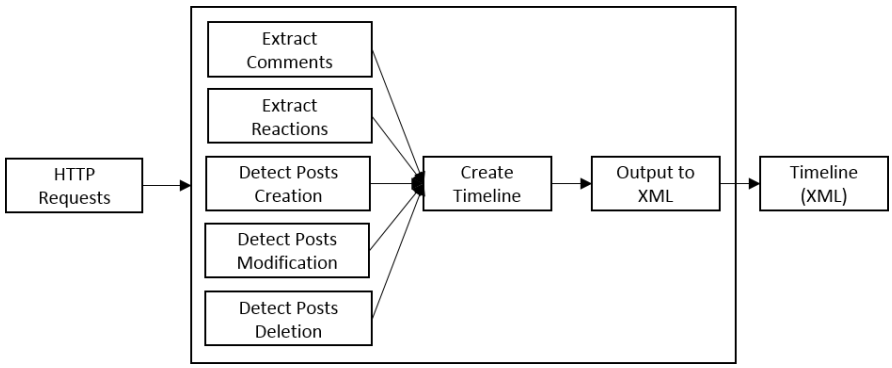

Figure 4.5: Social Network Mining Engine

4.5 Implementation

NetInfoMiner is currently implemented for Windows, but it can be straightforwardly extended to other OSes. The implementation includes four main engines, one for each targeted information type. The general implementation approach for each engine is shown in Figure 4.6.

NetInfoMiner leverage the various capabilities provided by *Wireshark* [39], in particular, the command line version *tshark*. *NetInfoMiner* receives as input the network traffic dump file and possibly the TLS key log file (KLF) or RSA private

key if available. Then, it initiates calls to *tshark* to extract the desired types of information by applying a sequence of *Wireshark* filters. A simplified syntax for all the applied filters for each mining engine are shown in table 4.3. The real filters are way too long and complex to present, and sometimes due to different column naming on *Wireshark*, some filters were split (i.e a filter for URL-encoded Form and another for json). Each filter takes care of keeping only packets needed in extracting a particular type of information (i.e. HTTP2 headers with Set-cookie values). The process of building these filters can be generalized as follows:

1. Record network traffic while performing the desired information to be extracted (login operation, visiting links, etc.)

2. Analyze the captured network traffic by looking for known keywords used in performing the operation. For instance, in credential extraction, search for the used credentials in the network traffic. This is used to identify the requests or responses that are used to carry the desired information

3. After identifying the requests or responses with the desired information, they are analyzed for common identifiable features, such as the protocol method, header names, parameter names, etc.

4. The identified features need to be incorporated into a filter to identify similar requests or responses in other network traffic.

The output of this filtering step is a sequence of strings representing only desired fields (time, source IP, etc.) from selected packets. In few cases, the

output of this step needs to be processed further due to some implementation limitations (e.g. HTTP2 data objects cannot be extracted in ASCII encoding directly). In this area, we depended heavily on the strong features provided by the task automation tool from Microsoft *PowerShell*. Afterwards, the processed results can be sent to the appropriate mining engine which is implemented in C language. The final output is collected from all mining engines to build an HTML formatted report for each victim.

Table 4.3: The Implemented Wireshark Filters for each Mining Engine in a Simplified Syntax

Mining Engine	**Simplified Wireshark Filters**	
HTTP Visited Links	(HTTP with Referer **OR** Request URI) **AND** HTTP with User Agent	
HTTP Credential	(URL-Encoded form **OR** Json) **AND** HTTP POST Method **AND** Full URI contains (log, auth, signin, signup, register, username, account,or password)	
HTTP2 Credential	(HTTP2 **OR** SPDY Request) **AND** data segment contains (pass, username, password, email, or session_,)	
HTTP Cookies	HTTP with Set-cookie	
HTTP2 Cookies	HTTP2 **OR** SPDY Header Name contains (set-cookie)	
Social Network (Facebook)	Comments	HTTP2 Or SPDY request to Facebook with data segment containing "comment_text="
	Reactions	HTTP2 Or SPDY request to Facebook with data segment containing "reaction_type="
	Add Post	HTTP2 Or SPDY request to Facebook with data segment containing "xhpc_message=“, "privacyx=“ and doesn’t contain "ref=edit"
	Edit Post	HTTP2 Or SPDY request to Facebook with data segment containing "xhpc_message=“, "privacyx=“ and "ref=edit"
	Delete Post	HTTP2 Or SPDY request to Facebook with header indicating “POST” method and URL containing “/ajax/timeline/delete?identifier=”

NetInfoMiner is designed with the support of HTTP, HTTP2 and SPDY protocols. There is a significant difference between the syntax of HTTP protocol in one hand and both HTTP2 and SPDY on another. For instance, in a single HTTP POST request, all the headers and data of the same request are close together

making it easier to extract and process them. However, in HTTP2 and SPDY, the requests and responses might be multiplexed and the data segment and the header segment of the same request might be interleaved with other segments of other requests. This makes the process of extracting the same information type (e.g. credentials) in HTTP2 and SPDY more complex than in HTTP. So, the processing performed by *NetInfoMiner* for HTTP2 and SPDY has a lot of differences and require further steps not required for HTTP. While the general approach shown in Figure 4.6 is still the same, but for the same targeted information type, totally different filters and output pre-processing are implemented. The extra processing required for HTTP2 and SPDY includes two parts. First, the current implementation of *Wireshark* (Version 2.2.1) does not provide well organized segmentation of the HTTP2 and SPDY headers as it does for HTTP. For instance, in HTTP, the $user - Agent$ header can be directly accessed by the column identifier *http.user_agent* while in HTTP2 and SPDY such identifier does not exist yet. The second extra processing is for having some required information type (i.e. credential parameter) in a different packet than the header segment of the same request, in case they were not transmitted together. This requires *NetInfoMiner* initiating an additional call for *tshark* to extract the required headers of a previously processed data segment.

A sample output of a browsing session is shown in Figure 4.7. The description of the XML fields in the extracted browsing sessions are mentioned below:

- Time: The time stamp of the first visited link included in the current brows-

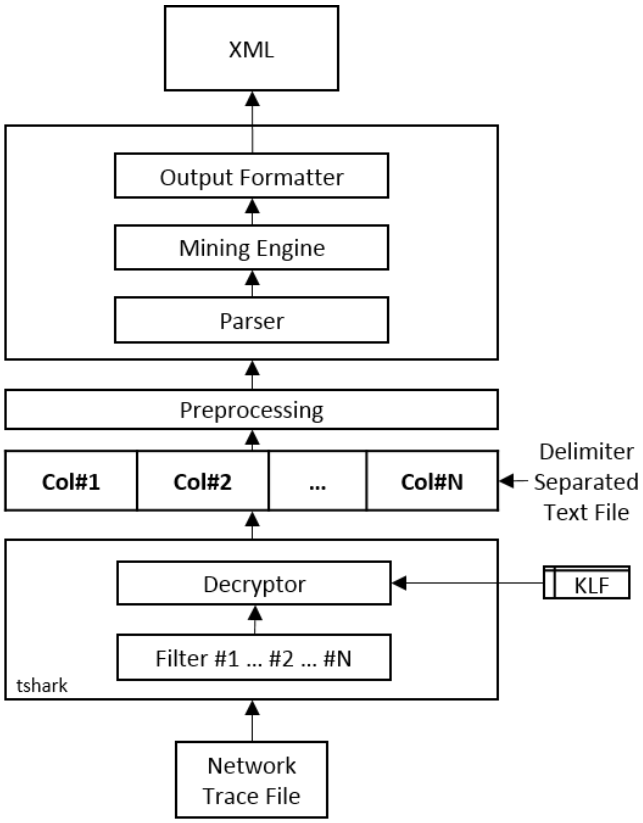

Figure 4.6: NetInfoMiner General Engine Implementation

ing session.

- Client: The IP address of the machine the client used to perform the current browsing session.

- Duration: The time between the first and last detected visited link in the current browsing session.

- Browser: The application from which the client conducted the current browsing session.

- visit_time: the time stamp of the visit of the link.

- link: the URI of the visited link

```
<BrowsingSession>
<time>Wed Sep 21 13:49:12 2016 </time>
<client>192.168.1.6</client>
<browser>Chrome 53.0.2785.116</browser>
<duration>+73 s</duration>
<Clicks>
<visit_time>Wed Sep 21 13:49:12 2016 </visit_time>
<link>https://myaccount.jawwal.ps/.../Url=%2f</link>
<visit_time>Wed Sep 21 13:49:32 2016 </visit_time>
<link>https://myaccount.jawwal.ps/.../InfoPage</link>
<visit_time>Wed Sep 21 13:50:25 2016 </visit_time>
<link>https://support.f5.com/.../sol12569.html</link>
</Clicks>
</BrowsingSession>
```

Figure 4.7: Extracted Browsing Session Sample

A sample extracted credential is shown in Figure 4.8. The description of the XML fields is mentioned below:

- Client: The IP address of the machine the client used to perform the login attempt.
- Host: The host name of the HTTP server that the client is trying to log into.
- Browser: The application from which the client used to perform the login attempt.
- LoginType: the type of the login process whether a Single-Page or Two-Page login.
- DetectionType: specifies the way the username and password were detected (Parameter naming or Heuristics).
- LoginTimeUsername: the time stamp when the username was initially sent

to the server.

- UsernameKey: the parameter name of the username.
- UsernameValue: the actual username string, might be an email.
- LoginTimePassword: the time stamp when the password was sent to the server. In Single-Page logins, the LoginTimeUsername and the LoginTimePassword will be exactly the same, since both parameters are sent together. While in Two-Page logins the timestamps will be different.
- PasswordKey: the parameter name of the password.
- PasswordValue: the actual password string.
- UsageCount: Number of times these credentials were used.
- LoginTime: the time stamp of beginning the login attempt into the above host with the above username and password. Since there might be several attempts.

For the cookies, all cookies exchanged between a client and a host will be represented in a different cookie set. A sample extracted cookie set is shown in Figure 4.9.

4.6 Extensibility

In this section, the extensibility feature in *NetInfoMiner* is highlighted. Also, some implemented features are hard coded in the current implementation but can

```
<Credential>
<Client>192.168.1.6</Client>
<Host>login.yahoo.com</Host>
<Browser>Torch 51.0.0.11603</Browser>
<LoginType>Two-Page Login</LoginType>
<DetectionType>Parameter Naming</DetectionType>
<LoginTimeUserName>DD MM dd hh:mm:ss YYYY</LoginTimeUserName>
<UsernameKey>username</UsernameKey>
<UsernameValue>*****</UsernameValue>
<LoginTimePassword>DD MM dd hh:mm:ss YYYY</LoginTimePassword>
<PasswordKey>passwd</PasswordKey>
<PasswordValue>*****</PasswordValue>
<UsageCount>1</UsageCount>
<UsageList>
<LoginTime>DD MM dd hh:mm:ss YYYY</LoginTime>
</UsageList>
</Credential>
```

Figure 4.8: Extracted Credential Sample

```
<CookieSet>
<Client>10.0.2.15</Client>
<Host>104.244.42.1</Host>
<Cookie>
<CookieFirstInitiationTime>
DD MM dd hh:mm:ss YY
</CookieFirstInitiationTime>
<CookieProperty>_twitter_sess=BAh...</CookieProperty>
<CookieProperty> Path=/</CookieProperty>
<CookieProperty> Domain=.twitter.com</CookieProperty>
<CookieProperty> Secure</CookieProperty>
<CookieProperty> HTTPOnly</CookieProperty>
</Cookie>
</CookieSet>
```

Figure 4.9: Extracted Cookie Set Sample

be modified to provide extensibility, these features are discussed with the required modifications to reach that goal. *NetInfoMiner* was designed with extensibility in mind, in order to make it a solution that can last longer. The main reasons that require the extensibility feature to exist is the following:

1. All the mining engines in *NetInfoMiner* depend on a group of *Wireshark* filters to extract raw traffic content with possible existence of desired information. Modifying these filters has a crucial effect on the accuracy of

information extraction. So, in the future, if the method to extract a desired information changes, these filters can be modified easily in text format and do not require any compilation. For instance, to extract possible login requests for the credential engine, all HTTP POST requests to a page that includes the phrase "login" are forwarded for further processing, if in the future a new common phrase is noticed in login pages, this phrase can be added to the *Wireshark* filter that is responsible for extracting the login requests.

2. The visited links engine depends on a list of known advertisement URLs used to filter out requests that are automatically generated for advertisement proposes. The list exist in a separate text file and can be manually edited to optimize the accuracy of the extraction in case new advertisement requests have been noticed and require to be filtered out.

3. The credential and session cookie engines depend on a hard coded list of parameter names (keywords) that most likely hold usernames and passwords (uname, passwd, etc.) or session cookies (ASPSESSION, PHPSESSION, etc.). These keywords might change in the future, this would affect the accuracy dramatically. To make the solution extensible, the code must be modified to read these keywords from a text file rather than being hard coded.

CHAPTER 5

EVALUATION AND RESULTS

5.1 Evaluation Limitations

In the process of evaluating *NetInfoMiner* the following limitations were faced:

1. There is no existing data set that can be used to evaluate the developed tool. The reasons for that is related to the following:

   (a) The tool can analyze encrypted traffic if the session keys were acquired. In this work, the session keys were acquired using the proposed DESSK attack, other network traffic encrypted using unavailable session keys cannot be analyzed using this tool.

   (b) To the best of our knowledge, there is no party that is willing to share network traffic that holds users credentials or that can expose users privacy.

2. The number of responding test subjects was limited due to the sensitivity

of the experiment (exposing credentials).

3. Performing comparative evaluation against similar tools is not possible due to different supported protocols and different extracted data types.

4. The evaluation of the visited links mining engine was limited and performed manually due to the lack of mechanism to collect the true values of visited links from test subjects.

5.2 Testing VM and Web Applications List

For the evaluation process, a windows VM was created and distributed among a group of test subjects. The VM included a prototype of the *SSLKeyStealer* malware, upon the launching of the malware the infection process starts, and the malware schedules a task to periodically transmit both the captured network traffic generated by the subjects in addition to the created KLF. To collect data generated by popular websites, we setup a local page including links to the top 500 websites according to Alexa. Each test subject was asked to visit some websites and create dummy accounts and then log into these accounts.

5.3 NetInfoMiner Tests and Results

After completing the data collection from the test subjects, *NetInfoMiner* was used for the information extraction.

5.3.1 Visited Links Results

The evaluation of the visited links mining engine was done by comparing the results with the browser history, the results show that *NetInfoMiner* successfully identifies all the visited links with few false positives resulted by uncontrolled redirects as mentioned in the original paper [21]. There is no quantitative measurements for the accuracy of the visited links mining engine due to the lack of mechanism to acquire them. Nonetheless, *NetInfoMiner* applies the RCI approach mentioned in [21] which is claimed to have an accuracy up to 95% of extracting the visited links. Also, the addition of filtering known advertisement requests as suggested in [18] claims extracting visited links with false positives between 0.74%-1.16% using the RCI approach. The only addition in *NetInfoMiner* is the support of HTTPS. Due to the reasons mentioned above, the accuracy of the visited links in *NetInfoMiner* can be assumed to be similar to the the accuracy mentioned in ClickMiner [18], until proven otherwise.

5.3.2 Credentials and Session Cookies Results

Regarding the evaluation of the credential and session cookie mining engines, each test subject reported how many websites he logged into. That information is then used to tell how many of them *NetInfoMiner* could extract based only on the test subject's network traffic.

Figure 5.1 shows the results of the empirical evaluation of the credential engine. Among 70 communicated credentials, *NetInfoMiner* could successfully identify 65

of them, that is 92.86%. We manually analyzed the traffic to understand the reason NetInfoMiner missed these 5 websites. We noticed no POST requests generated from these websites neither in HTTP, HTTP2, nor SPDY. We used the "Network recorder" feature available in most browser developer tools; Pinterest for instance, after a successful login generated a GET request with only the session cookie, the credentials was not observed anywhere. Other websites didn't generate any HTTP request on the network traffic while login.

Credential Engine					
Extraction	Tested Web Applications				
Successful	000webhost	dailymail	ieee	nytimes	storify
	about.me	dailymotion	instagram	oracle	teamviewer
	adobe	dell	imdb	paypal	ted
	ahlionline	dropbox	jawwal	portal.hadara.ps	theguardian
	alexa	easychair	jobs.uaeu.ac.ae	reddit	tumblr
	amazon	ebay	kfupm-office	riders.uber	twitter
	aol	edas	kickstarter	sedo	vimeo
	apple	etsy	kooora	sharelatex	weebly
	arab bank	facebook	linkedin	sonyplaystation	xing
	bbc	flickr	live	souq	yahoo
	bing	forbes	login.skype	sourceforge	yelp
	blogger	godaddy	meetuplogin	spotfire.cloud.tibco	youtube
	change.org	google	myspace	stackoverflaw	zdnet
Failed	aliexpress	booking	pinterest	tripadvisor	wikipedia
Total	70	Successful	65	Failed	5
				Percentage	92.86%

Figure 5.1: The web applications used in the evaluation of the credential engine

The results of the session cookie engine as shown in figure 5.2, shows that it is able to extract the session cookies from 77.14% of the tested web applications.

In the process of evaluating the credential and session cookie engines, many experiments were conducted that involves capturing login attempts into web application, only three of those were documented in details that enabled accurate measurements. Table 5.1 shows quantitative measurements of the accuracy of the credential engine when evaluated using three captured network traffic files with

Cookie Engine					
Extraction	Tested Web Applications				
Successful	000webhost	change.org	instagram	oracle	storify
	about.me	dailymail	jawwal	paypal	ted
	adobe	dropbox	jobs.uaeu.ac.ae	pintrest	tripadvisor
	ahlionline	ebay	kfupm-office	portal.hadara.ps	tumblr
	alexa	etsy	kooora	reddit	twitter
	aliexpress	facebook	linkedin	riders.uber	weebly
	amazon	flickr	live	sedo	wikipedia
	aol	godaddy	login.skype	sharelatex	yahoo
	arab bank	google	meetuplogin	sonyplaystation	youtube
	bbc	ieee	myspace	souq	zdnet
	blogger	imdb	nytimes	spotfire.cloud.tibco	
Failed	apple	dell	forbes	stackoverflaw	vimeo
	bing	easychair	kickstarter	teamviewer	xing
	booking	edas	sourceforge	theguardian	yelp
	dailymotion				
Total	70	Successful	54	Failed	16
				Percentage	77.14%

Figure 5.2: The web applications used in the evaluation of the cookie engine

login attempts to a group of widely famous web applications. The results display good accuracy in extracting login attempts with overall 6.35% false positive and 9.2% false negative. For instance, the third packet trace file (PCAP) captured 30 login attempts, 5 of them where not extracted by the credential engine and they are considered as false negatives, and 3 where not credentials and considered as false positives; they are parameters with names that match the expected parameter names of usernames and passwords. The overall percentage of the false positives was calculated for the three files as the ratio of the false positives compared with the total extracted login attempts. On the other hand, the overall percentage of the false negative was calculated for the three files as the ratio of the false negatives compared to the total number of actual login attempts.

Table 5.2 shows the quantitative measurements of the accuracy of the cookie engine for the same captured network traffic used for the evaluation of the credential engine. The results shows a significant lower accuracy compared to the

Table 5.1: Quantitative measurements of the accuracy of the credential engine

PCAP	Login Attempts	Extracted Login Attempts		FP	FN
		HTTP	HTTP2		
1	18	15	3	0	0
2	17	13	4	1	1
3	30	22	6	3	5
Total	65	50	13	4	6
			Inaccuracy	6.35%	9.23%

credential engine, with 40.04% false positive and 30.77% false negative. For instance, pcap file 3, with of 30 login attempts to 30 sites, the session cookie sets (cookies exchanged between client and server) of 13 sites were not extracted and considered as false negatives, and 5 extracted cookie sets were considered as false positives since they are related to known advertisements and content management web applications (adnxs.com, akamai.com). The overall percentage of the false positives was calculated for the three files as the ratio of the false positives compared with the total extracted session cookie sets. On the other hand, the overall percentage of the false negative was calculated for the three files as the ratio of the false negatives compared to the total number of login attempts. The results of the cookie engine suggest that it needs further improvements, especially targeting the elimination of the Ad related cookies which totaled 26.58% of the extracted session cookies as shown in Table 5.2.

Table 5.2: Quantitative measurements of the accuracy of the cookie engine

PCAP	Login Attempts	Extracted session cookie sets		FP	FN	Ad Cookies
		HTTP	HTTP2			
1	18	28	1	17	5	13
2	17	20	3	12	2	4
3	30	23	4	5	13	4
Total	65	71	8	34	20	21
			Inaccuracy	43.04%	30.77%	26.58%

5.3.3 Heuristics Engine Evaluation

In this section we provide the results of evaluating the heuristic engine which is a part of the credential engine. The reason why we had to provide an isolated evaluation of the heuristic engine is that most detected credentials are detected using parameter naming rather than heuristics. But, in the future, web applications may change their naming styles which would render the parameter naming detection approach insufficient. For that reason the heuristic engine is required to make *NetInfoMiner* a solution that could last longer. Unfortunately, there aren't much sites that wasn't detected using parameter naming. So, we resorted to evaluating the heuristic engine using the 10 million stolen usernames and passwords data set [51]. The goal was to evaluate the implemented heuristics and whether they can identify usernames and passwords correctly. Table 5.3 shows the results of testing the implemented heuristics and their ability to identify usernames and passwords. As the table shows, among 10 million passwords 94.8% were correctly identified as passwords but 67.95% were identified as usernames, this is related to some similar features between usernames and passwords that the implemented heuristics aren't able to correctly distinguish the difference. For instance, the username "mama1998mama" is also a possible password. Also, among the 10 million usernames, 91.66% was correctly identified as usernames but 89.86% was identified as passwords for the same earlier mentioned reason. On the other hand, the heuristics were tested on parameters that do not include any usernames or passwords. Among 6075 parameters, 26.5% were identified as passwords and 25.53%

were identified as usernames. For instance, the parameter en-US is a password according the implemented heuristics.

The above noticed results suggest that the heuristic engine still require more work due to significant amount of false detections. The additional work can be by either adding more distinctive heuristics or by optimizing the heuristic scores ranging.

Table 5.3: Results of testing the Heuristic scores on the 10M data set

	Identified as		Examples
	Password	Username	
Is Password (10M)	94.8	67.95	mama1998mama (98.66)
Is Username (10M)	89.86	91.66	kleiner.kruemmel86 (151.29)
Parameters (6075)	26.5	25.53	as pw: en-US (110.74)
			as un: treatment_general (103.29)

5.3.4 Social Network Results

For evaluating the extraction of Facebook common activities, the network traffic of performing the targeted activities were captured individually for several times, and all of them were extracted successfully.

For further evaluation, the following experiment were conducted:

1. In Firefox browser, the support for HTTP2 was disabled. The fallback protocol negotiated with Facebook if HTTP2 is not enabled was SPDY.

2. Logged into a Facebook account and performed the following activities:

   (a) Like PostID 10157703716340307

   (b) Comment "Hala Madrid!" to postID 10151132821914953

(c) Love postID 10157692692170307

(d) New Post "Hellooooooooo"

3. Switched to Chrome browser (HTTP2 is enabled by default) and performed the following activities:

(a) Edit postsID 10157814419505578 to "Hellooooooooo-Edit"

(b) Delete Post 10157814419505578

After capturing the traffic generated from performing the previous activities, NetInfoMiner was used to extract these activities from then traffic traces. The results show that, it was able to extract all of the previous activities. But, one of the activities was missing the user Facebook ID property which was expected to be extracted. This reflects that Facebook doesn't always use the same parameters for communicating the same activity. Nonetheless, there is a solution to recover that missing property by searching for it in the same recorded TCP stream in the network traffic. Facebook generates large amount of requests in a single TCP stream, a missing property in one request might exist in others. We manually inspected this option and found that the missing user Facebook ID exists in other requests in the same TCP stream of the recovered activity, and since a single TCP stream can only contain the activities of a single Facebook user, extracting that missing element from other requests in the same stream is a valid choice.

5.3.5 Disable/Enable HTTP2 Experiment

The effect of the negotiation of different protocols by clients and web applications is mentioned in Section (2.2). To evaluate how *NetInfoMiner* adopts with such effect, a quick experiment was conducted. Firefox browser enables the clients to disable or enable the support for HTTP2 protocol. We visited and logged in to the twitter website with HTTP2 enabled, then disabled the support for HTTP2 and re-logged again. We then analyzed the captured traffic of the experiment using *NetInfoMiner*. The tool successfully detected both login operations even though one was carried over HTTP2 while the other was over SPDY.

5.3.6 Ebay Special Case

A special case was observed in the ebay web application that demonstrated the advantage of the heuristic engine. As seen in figure 5.3, although there are parameters with names suggesting the existence of a username (userid) and a password (pass), the credential engine neglected these two parameters since they are empty. Instead, the heuristic engine detected both parameters correctly even though their names only includes numbers. The matching criteria for the username is its email syntax, and for the password it is its high heuristic score due to matching with enough number of password features to be assumed as one.

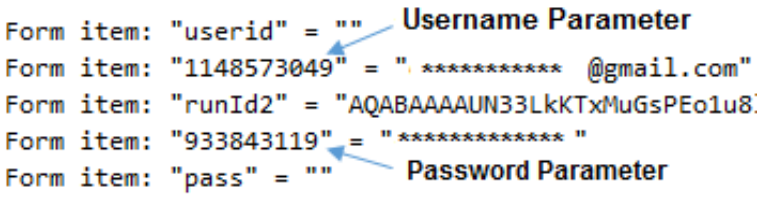

Figure 5.3: eBay Credential Parameters

5.4 Comparison

Table 5.4 shows a comparison of the capabilities of *NetInfoMiner* to other existing tools. *NetInfoMiner* targeted new types of information not targeted by existing solutions due to the new option of decrypting HTTPS traffic. As the table shows, *NetInfoMiner* is the only tool compared to the others that supports the analysis of HTTPS traffic. In addition to the support for HTTP2 and SPDY for extracting credentials, session cookies and Facebook common activities.

There is no need to compare the accuracy of the Visited links engine in *NetInfoMiner* against ClickMiner [18] since it applies the same RCI approach mentioned in [21] with the addition support for HTTPS protocol. Regarding NetworkMiner [19], to the best of our knowledge no measurable accuracy has been mentioned regarding the visited links extraction.

The credential engine in *NetInfoMiner* shows acceptable accuracy in extracting credentials carried across multiple protocols, HTTP, HTTPS, HTTP2 and SPDY. The credentials extraction in NetworkMiner has no mentioned accuracy, but, by testing it, it has very limited accuracy since it only dumps all HTTP POST parameters and cookies. Same thing regarding the session cookies, although *NetInfoMiner*'s session cookie engine didn't show the expected accuracy in extracting

session cookies, it still better than NetworkMiner, since the latter only dumps all cookies in the traffic in a separate tab with no fixation on session cookies.

Table 5.4: Supported features in NetInfoMiner tool compared to others

Extracted Info.	Protocol	Click-Miner	Network-Miner	NetInfo-Miner
Visited Links	HTTP	✓	✓	✓
	HTTPS	-	-	✓
	HTTP2 SPDY	-	-	In Progress
Credentials	HTTP	-	(*)	✓
	HTTPS	-	-	✓
	HTTP2 SPDY	-	-	✓
Session Cookies	HTTP	-	(*)	✓
	HTTPS	-	-	✓
	HTTP2 SPDY	-	-	✓
Facebook Activities	HTTP2	-	-	✓
(*) A dump of all cookies and parameters extracted from traffic without filtering undesired data.				

5.5 Learning Process

In this section we describe the process of improving the accuracy of *NetInfoMiner* and the possibility of making the leaning process automatic.

5.5.1 Manual Learning

Throughout the development of *NetInfoMiner* it was continuously evaluated using many network traffic captures to evaluate the extraction accuracy and perform improvements. The process of manual learning of *NetInfoMiner* can be summarized

as follows:

1. Traffic Capture Feeding: each attempt to teach *NetInfoMiner* starts with feeding it with a new traffic capture that contains the desired information to be extracted (Credential, cookies, etc).

2. Result Evaluation: after feeding *NetInfoMiner* with a traffic capture, the extraction results are manually inspected. *NetInfoMiner* was programmed to generate traces for each processing step to help in the learning process, these traces are manually inspected to figure out the reasons behind inaccurate results. The most common reasons behind inaccurate results were parsing issues or missing keyword (in case of credential extraction). Regarding the parsing issues, it is by far the most common issue behind inaccurate results; the parsing issues are related to the various techniques in web application development. For instance, some web applications applies an unconventional method for transporting the credentials using delimiters that conflict with delimiters used by *NetInfoMiner* in the different processing stages. Up to this point, *NetInfoMiner* breaks when processing the results for some web applications, it is minimized, rare, and can be circumvented manually, but it exits nonetheless. On the other hand, the missing keyword issues is at its minimum effect at this point due to the collection of adequate amount of keywords that can identify the desired information with acceptable accuracy.

3. Reprogramming: after identifying the reasons behind inaccurate results, the code of the related part of *NetInfoMiner* that is behind the inaccurate result

is modified to circumvent the issue. In most cases the modification involves changing a delimiter, adding a new keyword, or modifying *Wireshark* filter.

5.5.2 Automatic Learning

At this stage, automatic learning for *NetInfoMiner* is not implemented and is assumed difficult. But, we provide the requirements for making it possible. In general, providing a true positive source of information is the main key. For instance, in the credential engine, if the credential engine was somehow linked to the browser database for logged credentials to detect the usage of credentials. Then this information is used as a feedback to *NetInfoMiner*. Also the browser history can be used to improve the accuracy of the visited links engine.

CHAPTER 6

CONCLUSION

6.1 Summary of Findings

In this thesis, we proposed a new client side attack on SSL/TLS called DESSK. The attack exploits a new feature in some browsers that enables the logging of SSL/TLS session keys into a key log file (KLF) in plaintext. We implemented a prototype of a user-privilege malware called SSLKeyStealer which is responsible for the creation of the KLF and capturing network traffic if needed. The stolen keys can be later used in a network data mining tool developed by us called *NetInfoMiner*. It is responsible for the extraction of the desired information from the victim's network traffic. *NetInfoMiner* is composed of four data mining engines, the visited links engine, the credential engine, the session cookie engine, and the social network mining engine. All four engines support extracting information from HTTP and HTTPS traffic. In addition to the support for HTTP2 and SPDY in both the credential and cookie engines.

The visited links engine was not properly evaluated due to missing data set and source of true positive, but, it is based on an existing approach (RCI) that claims accuracy of extracting the visited links between 91-95%.

The evaluation of the credential engine shows that it is able to extract credentials from 65 out of 70 tested web applications. Further testing revealed that the credential engine can generate 6.35% false positives and miss 9.23% false negatives. These results suggest acceptable accuracy in extracting credentials. The false positives are related to some parameter names that match possible parameters known to hold usernames or passwords, while false negatives are related to either parsing issues or unknown implemented login criteria by some web applications

The cookie engine showed less accuracy compared to the credential engine. It was able to extract the session cookies from 54 of 70 tested web applications. Further testing revealed that the cookie engine can generate 43.04% false positives and miss 30.77% false negatives. These results suggest that more work needs to be done on the session cookie engine. The false positives are mostly related to some advertisements sites that creates session cookies to track users browsing activities. Testing showed that 26.58% of the extracted session cookies are related to advertisement purposes to some fixed sites such as adnxs.com. Possible solution can be 1) adding a list of known advertisement sites that can be used to filter out their cookies, 2) correlating the results of the session cookie engine with the credential engine, and only extract session cookies of sites that belong to extracted credentials in the same time frame.

The social network engine was implemented to extract common Facebook activities. It's testing shows that it can extract all targeted activities even if they are carried across multiple protocols and browsers. In some cases, some extracted activities cannot be related to a Facebook user ID due to a missing parameter. This issue has not been avoided in the current implementation but it might be overcome by looking for the existence of a Facebook user ID in the same TCP stream that the activity was extracted from.

At last, we believe that the key logging feature in its current implementation imposes the risk of exposing users sensitive information. The KLF should not be in plaintext.

6.2 Contributions

In this thesis we have two contributions:

1. NetInfoMiner: a network data mining tool able to extract different types of information carried across four different types of protocols, namely HTTP, HTTPS, HTTP2, and SPDY.

2. DESSK: a client side TLS attack, able to provide full decryption of TLS secure sessions by enabling the logging of session keys into a file on disk then stealing that file.

6.3 Future Work

The proposed *NetInfoMiner* is a part of an ongoing research. the shortcomings and possible future work are as follows:

1. Although it supports HTTP2 and SPDY for user credentials, and session cookies, the priority is to add the support for HTTP2 and SPDY protocols to the visited links engine

2. The credential and session cookie engines are programmed with hard coded keywords to identify possible usernames, passwords and cookies. To make them extensible, the code needs to be modified to look for these keywords from and external, modifiable look-up table.

3. The testing of the heuristic engine, as a part of the credential engine showed low accuracy due to insufficient heuristics. Currently, this doesn't affect the accuracy of the credential engine since most credentials are extracted using parameter naming, but, if it happens in the future that the parameter naming styles changes, improving the heuristic engine becomes a necessity.

4. The session cookie engine showed less accuracy compared to the credential engine due to the fixation on session cookie naming only. In addition to the need to filter cookies related to advertisement sites. Also, we intend to add support for other session management methods like URL rewriting.

5. The concept of user tracking though Facebook social plug-ins (i.e. Like button) is used to track real-life identities of users [54]. We plan to build on

this concept the ability of identifying user's real-life identities from cookies set by Facebook tracking system.

REFERENCES

[1] W. W. W. Consortium *et al.*, "Internet live stats."

[2] trustworthyinternet.org, "Trustworthy internet movement," https://www.trustworthyinternet.org/ssl-pulse/, accessed: October 26, 2016.

[3] Alexa.com, "imdb.com traffic statistics."

[4] InternetWorldStats.com, "World internet users statistics and 2015 world population stats. (n.d.)," http://www.internetworldstats.com/stats.htm , accessed: June 30, 2016.

[5] C. V. N. Index, "The zettabyte era–trends and analysis," *Cisco white paper*, 2013.

[6] S. Vandeven, "Ssl/tls: What?s under the hood," *SANS Institute InfoSec Reading Room*, vol. 13, 2013.

[7] E. Rescorla, *SSL and TLS: designing and building secure systems*. Addison-Wesley Reading, 2001, vol. 1.

[8] T. Dierks and C. Allen, "The tls protocol version 1.0," 1999.

[9] T. Dierks and E. Rescorla, "The transport layer security protocol version 1.1," RFC 4346, April, Tech. Rep., 2006.

[10] T. Dierks, "The transport layer security (tls) protocol version 1.2," 2008.

[11] J. Salowey, "Transport layer security (tls) session resumption without server-side state," *Transport*, 2008.

[12] J. Kambic, "Extracting cng tls/ssl artifacts from lsass memory."

[13] w3schools.co, "Browser statistics. (n.d.)," http://www.w3schools.com/browsers/browsers_stats.asp, accessed: April 20, 2016.

[14] mozilla.org, "Nss key log format. (n.d.)," https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Key_Log_Format, accessed: April 20, 2016.

[15] www.digicert.com, "Creating a .pem file for ssl certificate installations. (n.d.)," https://www.digicert.com/ssl-support/pem-ssl-creation.htm, accessed: April 20, 2016.

[16] A. Finamore and K. Papagiannaki, "Is the web http/2 yet?" in *Passive and Active Measurement: 17th International Conference, PAM 2016, Heraklion, Greece, March 31-April 1, 2016. Proceedings*, vol. 9631. Springer, 2016, p. 218.

[17] B. Patch, "Http/2 for a faster web," https://cascadingmedia.com/insites/2015/03/http-2.html.

[18] C. Neasbitt, R. Perdisci, K. Li, and T. Nelms, "Clickminer: Towards forensic reconstruction of user-browser interactions from network traces," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 1244–1255.

[19] E. Hjelmvik, "Passive network security analysis with networkminer," *IN) SECURE*, no. 18, pp. 1–100, 2008.

[20] P. Vadrevu, B. Rahbarinia, R. Perdisci, K. Li, and M. Antonakakis, "Measuring and detecting malware downloads in live network traffic," in *European Symposium on Research in Computer Security*. Springer, 2013, pp. 556–573.

[21] G. Xie, M. Iliofotou, T. Karagiannis, M. Faloutsos, and Y. Jin, "Resurf: Reconstructing web-surfing activity from network traffic," in *IFIP Networking Conference, 2013*. IEEE, 2013, pp. 1–9.

[22] A. Berger, A. D?Alconzo, W. N. Gansterer, and A. Pescapé, "Mining agile dns traffic using graph analysis for cybercrime detection," *Computer Networks*, vol. 100, pp. 28–44, 2016.

[23] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu, "Statistical identification of encrypted web browsing traffic," in *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*. IEEE, 2002, pp. 19–30.

[24] G. D. Bissias, M. Liberatore, D. Jensen, and B. N. Levine, "Privacy vulnerabilities in encrypted http streams," *Lecture notes in computer science*, vol. 3856, p. 1, 2006.

[25] R. Alshammari and A. N. Zincir-Heywood, "Machine learning based encrypted traffic classification: Identifying ssh and skype." *CISDA*, vol. 9, pp. 289–296, 2009.

[26] B. Miller, L. Huang, A. D. Joseph, and J. D. Tygar, "I know why you went to the clinic: risks and realization of https traffic analysis," in *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 2014, pp. 143–163.

[27] Y. Sheffer, R. Holz, and P. Saint-Andre, "Summarizing known attacks on transport layer security (tls) and datagram tls (dtls)," Tech. Rep., 2015.

[28] B. Taubmann, C. Frädrich, D. Dusold, and H. P. Reiser, "Tlskex: Harnessing virtual machine introspection for decrypting tls communication," *Digital Investigation*, vol. 16, pp. S114–S123, 2016.

[29] N. Perlroth, J. Larson, and S. Shane, "Nsa able to foil basic safeguards of privacy on web," *The New York Times*, vol. 5, 2013.

[30] P. G. Sarkar and S. Fitzgerald, "Attacks on ssl a comprehensive study of beast, crime, time, breach, lucky 13 & rc4 biases," *Internet: https://www. isecpartners. com/media/106031/ssl_attacks_survey. pdf [June, 2014]*, 2013.

[31] W. Alcorn, C. Frichot, and M. Orru, *The Browser Hacker's Handbook*. John Wiley & Sons, 2014.

[32] J. Hodges, C. Jackson, and A. Barth, "Http strict transport security (hsts)," Tech. Rep., 2012.

[33] E. H. News, "Rogue proxies hijack https traffic."

[34] Reddit.com, "Powershell 4.0 but no "new-selfsignedcertificate"?" https://www.reddit.com/r/PowerShell/comments/3190yr/powershell_40_but_no_newselfsignedcertificate/.

[35] D. Akhawe, B. Amann, M. Vallentin, and R. Sommer, "Here's my cert, so trust me, maybe?: understanding tls errors on the web," in *Proceedings of the 22nd international conference on World Wide Web*. ACM, 2013, pp. 59–70.

[36] K. Benton and T. Bross, "Timing analysis of ssl/tls man in the middle attacks," *arXiv preprint arXiv:1308.3559*, 2013.

[37] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni *et al.*, "Drown: Breaking tls using sslv2."

[38] J. Stüttgen and M. Cohen, "Robust linux memory acquisition with minimal target impact," *Digital Investigation*, vol. 11, pp. S112–S119, 2014.

[39] G. Combs *et al.*, "Wireshark," *Web page: http://www. wireshark. org/last modified*, pp. 12–02, 2007.

[40] Wikibooks.org, "Windows batch scripting," https://en.wikibooks.org/wiki/Windows$_{B}atch_{S}cripting$.

[41] B. Payette, *Windows PowerShell in action*. John Wiley & Sons, 2007.

[42] C. Laplace, M. Berg, H. Lai, and Y. Mandravellos, "Dev-c++ 4.9. 9.2," *Free Software Foundation Inc., http://www. bloodshed. net/, Cambridge, Massachusetts, USA*, 1991.

[43] curl.haxx.se, "curl: command line tool and library for transferring data with urls," https://curl.haxx.se/.

[44] Netresec.com, "Rawcap network sniffer." http://www.netresec.com/?page=RawCap.

[45] I. VirtualBox, "The virtualbox architecture," 2008.

[46] V. Jacobson, C. Leres, and S. McCanne, "The tcpdump manual page," *Lawrence Berkeley Laboratory, Berkeley, CA*, 1989.

[47] Juniper.net, "How many packets per second per port are needed to achieve wire-speed?" https://kb.juniper.net/InfoCenter/index?page=contentid=KB14737actp=search.

[48] w3techs.com, "Usage of http/2 for websites," https://w3techs.com/technologies/details/ce-http2/all/all, accessed: October 20, 2016.

[49] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang, "The tangled web of password reuse." in *NDSS*, vol. 14, 2014, pp. 23–26.

[50] B. Ur, F. Noma, J. Bees, S. M. Segreti, R. Shay, L. Bauer, N. Christin, and L. F. Cranor, "" i added'!'at the end to make it secure": Observing password creation in the lab," in *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, 2015, pp. 123–140.

[51] wpengine.com, "Unmasked: What 10 million passwords reveal about the people who choose them," http://wpengine.com/unmasked/, accessed: October 20, 2016.

[52] R. Siles, "Session management cheat sheet?session id properties," *Online at https://www. owasp. org/index. php/Session_ Management_ Cheat_ Sheet# Session_ ID_ Properties*, 2013.

[53] trends.builtwith.com, "Statistics for websites using framework technologies," https://trends.builtwith.com/framework, accessed: October 20, 2016.

[54] G. Acar, B. Van Alsenoy, F. Piessens, C. Diaz, and B. Preneel, "Facebook tracking through social plug-ins," 2015.

Vitae

- Name: Ahmad Amro
- Nationality: Palestinian
- Date of Birth: November 18th, 1986
- Email: *engahmadamro@{live.com,gmail.com}*
- Permanent Address: Dura, Hebron, West-bank, Palestine

- Bachelor of Engineering (B.E.), Computer Systems Engineering, Palestine Polytechnic University, Palestine, 2010.
- Published papers:
    1. "Known-Plaintext Attack and Improvement of PRNG-Based Text Encryption" in 2016 7th International Conference on Information and Communication Systems (ICICS).
    2. "NetInfoMiner: High-level Information Extraction From Network Traffic" in 2017 IEEE International Conference on Big Data and Smart Computing.
    3. "Double-Hashing Operation Mode for Encryption" in 2017 IEEE 7th Annual Computing and Communication Workshop and Conference.